DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529-0162

# High-performance Monitoring Architecture for Large-scale Distributed Systems Using Event Filtering

**By**
Dr. K. Maly
Department of Computer Science

IN-61
348

## FINAL REPORT
**For the period ending August 15, 1998**

**Prepared for**
NASA Langley Research Center
Attn.: Mr. Joseph Murray
Grant Officer
Mail Stop 126
Hampton, VA 23681-0001

And

NASA Langley Research Center
Attn.: Mr. Robert. W. Wills
Technical Officer
Mail Stop 152-D
Hampton, VA 23681-0001

**Under**
NASA Grant NAG 1-908 - Supplement No. 16
ODURF Project No. 187264

**October 1998**

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529-0162

# High-performance Monitoring Architecture for Large-scale Distributed Systems Using Event Filtering

**By**
Dr. K. Maly
Department of Computer Science

## FINAL REPORT

**For the period ending August 15, 1998**

**Prepared for**
NASA Langley Research Center
Attn.: Mr. Joseph Murray
Grant Officer
Mail Stop 126
Hampton, VA 23681-0001

And

NASA Langley Research Center
Attn.: Mr. Robert. W. Wills
Technical Officer
Mail Stop 152-D
Hampton, VA 23681-0001

**Under**
NASA Grant NAG 1-908 - Supplement No. 16
ODURF Project No. 187264

**Submitted by**
Old Dominion University Research Foundation
800 West 46th Street
Norfolk, VA 23508

**October 1998**

# Designing a High Speed Network: An Application-oriented Approach

K. Maly

Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

*Email: maly@cs.odu.edu*
*(757) 683-4817*
*Fax: (757) 683-4900*

## Final Report

## Submitted to:

Wayne Bryant, FETD
Assistant Division Chair
NAG-1-908, ODURF Project No. 187264

National Aeronautics and Space Administration
Research Support Contracts Branch
Langley Research Center - Mail Stop 126
Hampton, VA 23681-2199

# High-performance Monitoring Architecture for Large-scale Distributed Systems Using Event Filtering

Ehab S. Al-Shaer
Department of Computer Science
Old Dominion University
Norfolk, Virginia, 23529-0162
ehab@cs.odu.edu

## Abstract

Monitoring is an essential process to observe and improve the reliability and the performance of large-scale distributed (LSD) systems. In an LSD environment, a large number of events is generated by the system components during its execution or interaction with external objects (e.g. users or processes). Monitoring such events is necessary for observing the run-time behavior of LSD systems and providing status information required for debugging, tuning and managing such applications. However, correlated events are generated concurrently and could be distributed in various locations in the applications environment which complicates the management decisions process and thereby makes monitoring LSD systems an intricate task.

We propose a scalable high-performance monitoring architecture for LSD systems to detect and classify interesting local and global events and disseminate the monitoring information to the corresponding end-points management applications such as debugging and reactive control tools to improve the application performance and reliability. A large volume of events may be generated due to the extensive demands of the monitoring applications and the high interaction of LSD systems. The monitoring architecture employs a high-performance event filtering mechanism to efficiently process the large volume of event traffic generated by LSD systems and minimize the intrusiveness of the monitoring process by reducing the event traffic flow in the system and distributing the monitoring computation. Our architecture also supports dynamic and flexible reconfiguration of the monitoring mechanism via its instrumentation and subscription components. As a case study, we show how our monitoring architecture can be utilized to improve the reliability and the performance of the Interactive Remote Instruction (IRI) system which is a large-scale distributed system for collaborative distance learning.

The filtering mechanism represents an intrinsic component integrated with the monitoring architecture to reduce the volume of event traffic flow in the system, and thereby reduce the intrusiveness of the monitoring process.

We are developing an event filtering architecture to efficiently process the large volume of event traffic generated by LSD systems (such as distributed interactive applications). This filtering architecture is used to monitor collaborative distance learning application for obtaining debugging and feedback information. Our architecture supports the dynamic (re)configuration and optimization of event filters in large-scale distributed systems. Our work represents a major contribution by (1) survey and evaluating existing event filtering mechanisms in supporting monitoring LSD systems and (2) devising an integrated scalable high-performance architecture of event filtering that spans several key application domains, presenting techniques to improve the functionality, performance and scalability. This paper describes the primary characteristics and challenges of developing high-performance event filtering for monitoring LSD systems. We survey existing event filtering mechanisms and explain key characteristics for each technique. In addition, we discuss limitations with existing event filtering mechanisms and outline how our architecture will improve key aspects of event filtering.

## Summary

The demands of large-scale distributed (LSD) systems is increasing. There are two main influential factors that encourage employing such applications in many domains: the advances in the Internet and the Intranets technology and the economical and performance benefits of distributed applications, in general. Examples of LSD systems include large-scale collaborative distance learning, video conferencing, group-ware editing, distributed transaction systems and distributed interactive simulation. LSD systems involve a large number of users or application entities which are geographically dispersed over interconnected LANs (i.e. Intranets) or over WAN (e.i. Internet). Such applications enable more interaction and resource sharing that go beyond the limitation of long geographical distances. For instance, in large-scale distance learning and training applications, a large number of students who may be far distant from each other can share the same benefit of attending a course, and exchanged experiences regardless of the distance or the number of participants. Similarly, database systems (such as banking systems) can increase its computational and data storage capacity by utilizing large-scale transaction systems.

Due to the distributed nature and the large number of participants or application entities in LSD system, reliability and performance of such applications become critical issues. The wide geographical distribution and large interaction of such applications may sometimes increase the possibility of failures/errors or performance bottlenecks. Unlike centralized or isolated applications, LSD systems are likely to deal with different environments that may be dynamically changing. For these reasons, observing the run-time behavior of LSD systems is essential to discover reliability and performance problems and initiate the proper reactions to alleviate these problems. These actions are performed either at run-time such as fault recovery and applications steering or at development-time such as fixing bugs and design enhancements.

The LSD system developers or managers (could be human or software components) require feedback information on the system behavior for testing and debugging purposes or for fault recovery and performance tuning procedures. Monitoring LSD systems is an effective means to observe applications at run-time and provide this feedback information to the system developers and system managers in order to to improve reliability, robustness and performance of LSD systems.

In LSD system, a large number of events is generated by the system components (e.g. processes) during its execution or interaction with an external objects such as users or other applications. These events represent the run-time behavior of LSD systems. In centralized or isolated systems, developers express these events via "print" statements or via using any generic debugger tool (e.g. gdb) for monitoring and inspecting the application behavior. However, in LSD systems, monitoring is much more complex because events can be concurrent and distributed in the application environment. In other words, unlike centralized or isolated systems, events that may represent application errors or bugs may be correlated and distributed over many locations. For example, application errors related to the communication operations obviously involve observing the sender(s) and the receivers(s). Similarly, the knowledge of application performance is also distributed in the application environment. For instance, calculating the average load of the system must involve all participant machines. Thus, concurrency and distribution of events makes the tasks of testing and management decisions much harder.

Unlike centralized or isolated systems, in LSD environment, errors/failures events (due to software bugs or improper implementation) or events that convey the application status can be dispersed over many different locations and application entities which makes the task of testing and debugging or management decisions process (e.g. fault recovery or performance tuning procedures) much harder to achieve in LSD systems.

Furthermore, the monitored events, either simple (local) or complex (global) events, which may encounter network latency and clock skewing problems. The large volume of event traffic which flows in the system may swamp the monitoring process.

In this thesis, we present a scalable high-performance architecture for monitoring interesting local and global events of LSD systems and disseminating the monitoring information dynamically to the subscribing monitoring applications such as distributed debugging and reactive control tools which subsequently would effectively improve the robustness (via debugging), the reliability (via fault recovery), and the performance (via performance tuning and application steering). Although our main focus in this proposal is on these monitoring applications: reliability and performance tuning, the proposed monitoring architecture can be

used for many other monitoring applications such as security and correctness checking. We show how the design can satisfy the work objectives of supporting a scalable, high-performance, dynamic, flexible and non-intrusive architecture for monitoring large-scale distributed systems. Various optimization techniques are proposed for the event filtering mechanism to increase the scalability and the performance of the monitoring process and minimize overhead on the computation and the network resources which in turn reduces the intrusiveness of the monitoring operations. We show an emphasis in studying and designing the filtering component of the monitoring architecture, since the event filtering mechanism is an intrinsic component that has a significant impact on the monitoring performance and scalability. Our monitoring architecture provides a simple mechanism to prepare (i.e. instrument) the monitored applications for capturing and reporting generated events with a minimal intervention from the application developers. It also supports a dynamic monitoring by enabling the monitoring applications (e.g. managers) to dynamically change their monitoring requests (called subscription) at run-time. The monitoring architecture provide a flexible service by providing priority-based event monitoring service where events are processed based on their priorities and adjustable instrumentation mechanism (e.g. adjusting the event reporting rate).

Although there is a considerable amount of research work on monitoring in general and monitoring distributed and parallel applications in specific, the proposed systems are insufficient for satisfying our design objectives and supporting a scalable, high-performance, dynamic, flexible and non-intrusive monitoring architecture for large-scale distributed systems.

## Debugging Example in Monitoring IRI Application

In this section, we show an example of using the monitoring architecture for testing and debugging IRI application. The goal of this example is to illustrate more the monitoring process and show how our monitoring architecture can be used effectively and easily for testing and debugging purposes. Many other examples of this application or other monitoring applications can be developed by following the a similar procedure.

The IRI application is collaborative distributed applications where exchanging messages is a major activity in the system. The Reliable Multicast Protocol Server (RMPS) is the main communication module in IRI application. In such message-based applications, it is very likely to encounter a mismatch in message sizes because of errors in sending/receiving operations such as type mismatch between the sender and the receiver(s) or the kernel alignment of sent packet[gif] . Therefore, debugging "send" and "receive" operations is highly desired in such message-based distributed application such as IRI as well as any client/server application, in general. Thus, our debugging example is monitoring the send and receive activities (events) in RMPS and reporting information about any mismatch in sent and received message sizes. In this monitoring example, the consumer(s) could be the developer(s) and RMPS entities are the event producers. This example represent a composite event since the knowledge of sent/received message sizes is distributed in the IRI virtual classroom. In the following, we describe how this monitoring example is constructed and processed in IRI application environment.

Filter Subscription in HFSL: A developer may define his/her filter subscription as follows:

FILTER= [tex2html_wrap_inline2827]
[(MSend.IPsrc = [tex2html_wrap_inline2831] = [tex2html_wrap_inline2833]
[tex2html_wrap_inline2835]
The MSend and MRec are the multicast send and receive events respectively. The Report_Mismatch action is a function or program that will send the monitoring information to the developer reporting the occurrence of this event.

Event Specifications in HESL: The send multicast event (called MSend) and the received multicast event (called MRec) may be specified respectively as follows:

EVENT= [tex2html_wrap_inline2837] Module_Name=RMPS, Func_Name=Send, NULL, Immediate;

IPdest=224.*.*.*, IPsrc=ANY [tex2html_wrap_inline2839] MSend.

EVENT= [tex2html_wrap_inline2837] Module_Name=RMPS, Func_Name=Receive, NULL, Immediate;

IPdest=224.*.*.*, IPsrc=ANY [tex2html_wrap_inline2839] MRec.

RMPS Instrumentation: The RMPS send and receive routines are instrumented to report information about any send and receive events according to the event specifications: event name, message sequence number, IP source address and size of sent and received message. These fields are used in the filter definition (program).

This shows the filtering internal representation after the filter is decomposed and distributed between the monitoring agents. The filter is decomposed to three subfilters: F1 which detects receiving multicast events and forwards it to its DMA, F2 which detects sending multicast events and forwards them to all DMAs, F3 which is responsible for evaluation the filter expression by comparing the receiving and the sending multicast events. If the composite event represented by Msg_Mismatch_FILTER is detected, then F3 will forward the monitoring information to the developer(s) as requested. Notice that sending and receiving events can take a place at any machine (RMPS resides on every machine in IRI application). Therefore, based on the Environment Specifications, the F1 and F2 subfilters are delegated to all LMAs in the virtual classroom. However, F3 are delegated to the DMAs which get the MSend and MRec events and evaluate the filter expression accordingly. The extracting layer in th figure is just to forward only the relevant information and reduce the event traffic. Finally, the requested monitoring information is forwarded to one or more developers based on their subscription. This simple example shows how a developer can effectively define and monitor IRI application activities at run-time and collect the desired debugging information from various locations in the application environment without the hassle of analyzing multiple traces or inspecting the application entities at different locations.

## Publications


"Interactive Distance Learning over Intranets", (with H. Abdel-Wahab, C.M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica, E. Al-Shaer and R. Talla), IEEE Journal of Internet Computing, Vol. 1, 1, Jan. 97, pp. 60-71

"Virtual Classrooms and Interactive Remote Instruction" (with C. Wild, C.M. Overstreet, H. Abdel-Wahab, A. Gupta, A. Youssef, E. Stoica, R. Talla, A. Prabhu), International Journal of Innovations in Education and Training, Vol.34, 1, pp. 44-51, 1997

Multimedia Modeling, World Scientific Publ. Co, Nov. 96, Edited by J. P. Courtiat, M. Diaz, P. Senac: Multimedia Integration into a Distance Learning Environment (with H. Abdel-Wahab, E. Stoica), pp. 69-84, see also II,74

"The Role of Multicasting in Interactive Multimedia Distance Learning Systems", (with H. Abdel-Wahab, E. Stoica and A. Youssef), Journal of Network and Systems Management: Special Issue on Multimedia Network/Service Management, Vol.3, No. 5, 1997

" Web Based Framework for Parallel Computing" , (with Z. Chen, P. Mehrotra, P.K. Vangala, M. Zubair), Journal of Concurrency Practice and Experience, Wiley , Vol 9, Nov. 1997

"Use of Web technology for interactive remote instruction", (with C. M. Overstreet, A. Gonzalez, M. Denbar, R. Cutaran, N. Karunatrane, C. J. Srinivas), Computer Networks and ISDN Systems, Vol. 30, 1-7, pp. 660-662, April 1998

"Adaptive Object-Oriented Filtering Framework for Event Management Applications", (with Ehab Al-Shaer, Mohamed Fayad, Hussein Abdel-Wahab), to appear in Journal of ACM Computing Survey, December, 1998